CSE 2001: Introduction to Theory of Computation

Summer 2013

Week 9: Turing Machines and the Church-Turing Thesis

Yves Lespérance

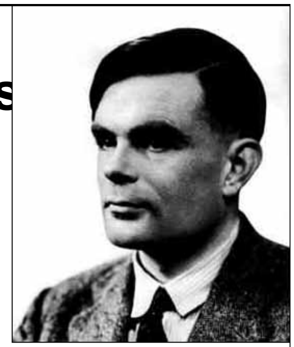
Course page: http://www.cse.yorku.ca/course/2001

Slides are mostly taken from Suprakash Datta's for Winter 2013

Next

- Computability (Ch 3)
  - Turing machines
  - TM-computable/recognizable languages
  - Variants of TMs

Turing Machines



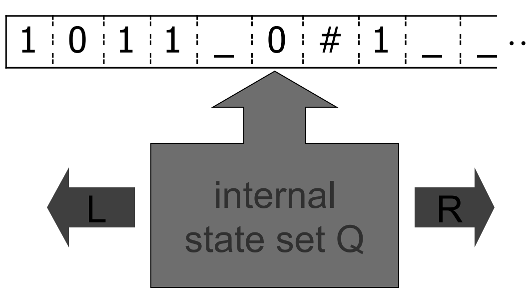
After Alan M. Turing (1912–1954)

In 1936, Turing introduced his abstract model for computation in his article "*On Computable Numbers, with an application to the Entscheidungsproblem*".

At the same time, Alonzo Church published similar ideas and results.

However, the Turing model has become the standard model in theoretical computer science.

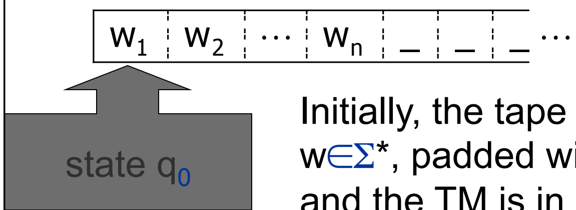
Informal Description TM



At every step, the head of the TM M reads a letter x_i from the one-way infinite tape.

Depending on its state and the letter x_i, the TM
- writes down a letter,
- moves its read/write head left or right, and
- jumps to a new state.

Input Convention

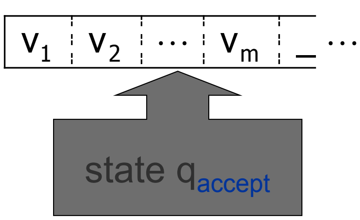


Initially, the tape contains the input $w \in \Sigma^*$, padded with blanks "_", and the TM is in start state q_0.

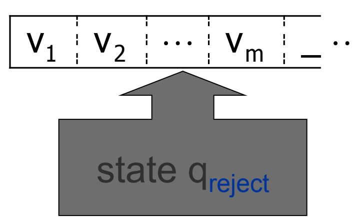
During the computation, the head moves left and right (but not beyond the leftmost point), the internal state of the machine changes, and the content of the tape is rewritten.

Output Convention

The computation can proceed indefinitely, or the machines reaches one of the two halting states:



or



Major differences with FA, PDA

- Input can be read more than once
- Scratch memory available, can be accessed without restrictions
- The "running time" is not predictable from the input – the machine can "churn" for a long time even on a short input
- So we need a clear indicator of end of computation

Turing Machine (Def. 3.3)

A Turing machine M is defined by a 7-tuple $(Q,\Sigma,\Gamma,\delta,q_0,q_{accept},q_{reject})$, with

- Q finite set of states
- Σ finite input alphabet (without "_")
- Γ finite tape alphabet with $\{ \_ \} \cup \Sigma \subseteq \Gamma$
- q_0 start state $\in Q$
- q_{accept} accept state $\in Q$
- q_{reject} reject state $\in Q$
- δ the transition function

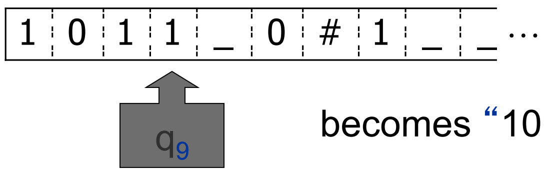
  $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L,R\}$

Why do you need these?

Configuration of a TM

The configuration of a Turing machine consists of
- the current state $q \in Q$
- the current tape contents $\in \Gamma^*$
- the current head location $\in \{0,1,2,\ldots\}$

This can be expressed as an element of $\Gamma^* \times Q \times \Gamma^*$:



becomes "101 q_9 1_0#1"

An Elementary TM Step

Let $u,v \in \Gamma^*$; $a,b,c \in \Gamma$; $q_i,q_j \in Q$, and M a TM with transition function δ.
We say that the configuration "ua q_i bv" <u>yields</u> the configuration "uac q_j v" if and only if:
$\delta(q_i,b) = (q_j,c,R)$.

Similarly, "ua q_i bv" yields "u q_j acv" if and only if
$\delta(q_i,b) = (q_j,c,L)$.

Also special cases for when head is at either end of the configuration; see Sipser for details.

Terminology

starting configuration on input w: "q_0w"

accepting configuration: "$uq_{accept}v$"

rejecting configuration: "$uq_{reject}v$"

The accepting and rejecting configurations are the *halting configurations*.

Accepting TMs

A Turing machine M <u>accepts</u> input $w \in \Sigma^*$
if and only if there is a finite sequence of configurations $C_1, C_2, \ldots, C_k$ with

- C_1 the starting configuration "q_0w"
- for all $i=1,\ldots,k-1$ C_i yields C_{i+1} (following M's δ)
- C_k is an accepting configuration "$uq_{accept}v$"

The language that consists of all inputs that are accepted by M is denoted by L(M).

Turing Recognizable (Def. 3.5)

A language L is Turing-recognizable if and only if there is a TM M such that L=L(M).

Also called: a recursively enumerable language.

Note: On an input $w \notin L$, the machine M can halt in a rejecting state, or it can 'loop' indefinitely.

How do you distinguish between a very long computation and one that will never halt?

Turing Decidable (Def. 3.6)

A language L=L(M) is decided by the TM M if on every w, the TM finishes in a halting configuration. (That is: q_{accept} for $w \in L$ and q_{reject} for all $w \notin L$.)

A language L is Turing-decidable if and only if there is a TM M that decides L.

Also called: a recursive language.

Example 3.7: A = { 0^j | $j=2^n$ }

Approach: If j=0 then “reject”; If j=1 then “accept”; if j is even then divide by two; if j is odd and >1 then “reject”. Repeat if necessary.

1. Sweep left to right crossing off every other zero.
   1. If the tape has a single 0, accept.
   2. Else If there are an odd number of zeros reject.
2. Return the head to the left-hand end of the tape.
3. goto 1

State diagrams of TMs

Like with PDA, we can represent Turing machines by (elaborate) diagrams.

See Figures 3.8 and 3.10 for two examples.

If transition rule says: $\delta(q_i,b) = (q_j,c,R)$, then:

$$q_i \xrightarrow{b \to c,R} q_j$$

When Describing TMs

It is assumed that you are familiar with TMs and with programming computers.

Clarity above all: high level description of TMs is allowed but should not be used as a trick to hide the important details of the program.

Standard tools: Expanding the alphabet with separator "#", and underlined symbols $\underline{0}$, $\underline{a}$, to indicate 'activity'. Typical: Γ = { 0,1,#,_,$\underline{0}$,$\underline{1}$ }

Some more examples

- B={w#w| w $\in$ (0,1)* } (Pg 172)

- C = {a^i b^j c^k | i*j=k, i,j,k >= 1} (Pg 174)

Turing machine variants

- Multiple tapes
- 2-way infinite tapes
- Non-deterministic TMs

Multitape Turing Machines

A k-tape Turing machine M has k different tapes and read/write heads. It is thus defined by the 7-tuple $(Q,\Sigma,\Gamma,\delta,q_0,q_{accept},q_{reject})$, with

- Q finite set of states
- Σ finite input alphabet (without "_")
- Γ finite tape alphabet with $\{\_\} \cup \Sigma \subseteq \Gamma$
- q_0 start state $\in Q$
- q_{accept} accept state $\in Q$
- q_{reject} reject state $\in Q$
- δ the transition function

$$\delta: Q\setminus\{q_{accept},q_{reject}\} \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L,R\}^k$$

k-tape TMs versus 1-tape TMs

Theorem 3.13: For every multi-tape TM M, there is a single-tape TM M' such that L(M)=L(M').
Or, for every multi-tape TM M, there is an equivalent single-tape TM M'.

Proving and understanding these kinds of robustness results, is essential for appreciating the power of the Turing machine model.

From this theorem Corollary 3.15 follows:
A language L is TM-recognizable if and only if some multi-tape TM recognizes L.

Outline Proof Thm. 3.13

Let $M=(Q,\Sigma,\Gamma,\delta,q_0,q_{accept},q_{reject})$ be a k-tape TM.
Construct 1-tape M' with expanded $\Gamma' = \Gamma \cup \underline{\Gamma} \cup \{\#\}$

Represent M-configuration

$$u_1 q_j a_1 v_1, \quad u_2 q_j a_2 v_2, \quad \ldots, \quad u_k q_j a_k v_k$$

by M' configuration

$$q_j \# u_1 \underline{a}_1 v_1 \# u_2 \underline{a}_2 v_2 \# \ldots \# u_k \underline{a}_k v_k$$

(The tapes are separated by #, the head positions are marked by underlined letters.)

Proof Thm. 3.13 (cont.)

On input $w=w_1\ldots w_n$, the TM M' does the following:

- Prepare initial string: $\#\underline{w_1}\ldots w_n\#\underline{\_}\#\cdots\#\underline{\_}\#\_\ \cdots$
- Read the underlined input letters $\in \Gamma^k$
- Simulate M by updating the input and the underlining of the head-positions.
- Repeat 2-3 until M has reached a halting state
- Halt accordingly.

PS: If the update requires overwriting a # symbol, then shift the part $\# \cdots \_$ one position to the right.

Non-deterministic TMs

A <u>nondeterministic Turing machine</u> M can have several options at every step. It is defined by the 7-tuple $(Q,\Sigma,\Gamma,\delta,q_0,q_{accept},q_{reject})$, with

- Q finite set of states
- Σ finite input alphabet (without "_")
- Γ finite tape alphabet with $\{\_\} \cup \Sigma \subseteq \Gamma$
- q_0 start state $\in Q$
- q_{accept} accept state $\in Q$
- q_{reject} reject state $\in Q$
- δ the transition function

$$\delta: Q\backslash\{q_{accept},q_{reject}\} \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L,R\})$$

Robustness

Just like k-tape TMs, nondeterministic Turing machines are not more powerful than simple TMs:

Every nondeterministic TM has an equivalent 3-tape Turing machine, which –in turn– has an equivalent 1-tape Turing machine.

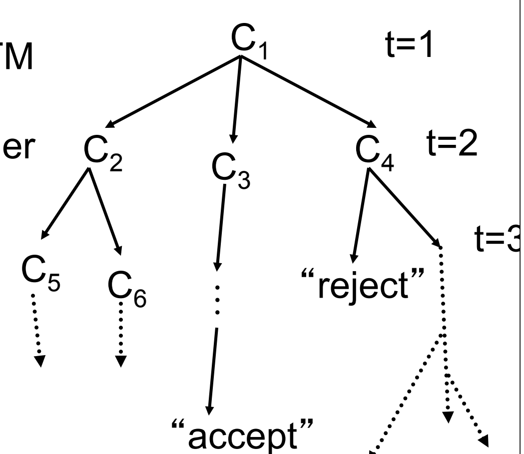
Hence: "A language L is recognizable if and only if some nondeterministic TM recognizes it."

The Turing machine model is extremely robust.

Computing with non-deterministic TMs

Evolution of the n.d. TM represented by a tree of configurations (rather than a single path).

If there is (at least) one accepting leave, then the TM accepts.



Simulating Non-deterministic TMs with Deterministic Ones

We want to search every path down the tree for accepting configurations.

Bad idea: "depth first". This approach can get lost in never-halting paths.

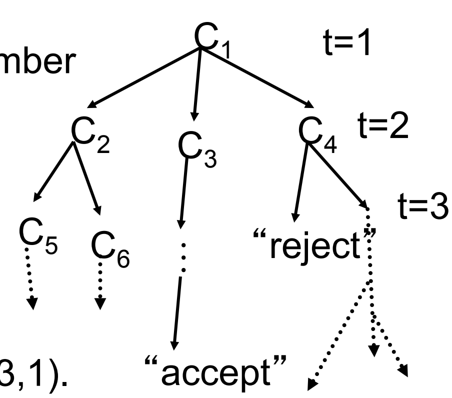
Good idea: "breadth first". For time step 1,2,… we list all possible configurations of the non-deterministic TM. The simulating TM accepts when it lists an accepting configuration.

Breadth First

Let b be the maximum number of children of a node.

Any node in the tree can be uniquely identified by a string $\in \{1,\ldots,b\}^*$.

Example: location of the rejecting configuration is (3,1).



With the lexicographical listing ε, (1), (2),…, (b), (1,1), (1,2),…,(1,b), (2,1),… et cetera, we cover all nodes.

Proof of Theorem 3.16

Let M be the non-deterministic TM on input w.

The simulating TM uses three tapes:
T1 contains the input w
T2 the tape content of M on w at a node
T3 describes a node in the tree of M on w.

1) T1 contains w, T2 and T3 are empty
2) Simulate M on w via the deterministic path to the node of tape 3. If the node accepts, "accept", otherwise go to 3)
3) Increase the node value on T3; go to 2)

Robustness

Just like k-tape TMs, nondeterministic Turing machines are not more powerful than simple TMs:

Every nondeterministic TM has an equivalent 3-tape Turing machine, which –in turn– has an equivalent 1-tape Turing machine.

Hence: "A language L is recognizable if and only if some nondeterministic TM recognizes it."

Let's consider other ways of computing a language…

Enumerating Languages

Thus far, the Turing machines were 'recognizers'.

When a TM E generates the words of a language, E is an <u>enumerator</u> (cf. "recursively enumerable").

A Turing machine E, <u>enumerates</u> the language L if it prints an (infinite) list of strings on the tape such that all elements of L will appear on the tape, and all strings on the tape are elements of L. (E starts on an empty input tape. The strings can appear in any order; repetition is allowed.)

Enumerating = Recognizing

Theorem 3.21: A language L is TM-recognizable if and only if L is enumerable.

<u>Proof</u>: ("if") Take the enumerator E and input w.
Run E and check the strings it generates.
If w is produced, then "accept" and stop, otherwise let E continue.
("only if") Take the recognizer M. Let $s_1,s_2,\ldots$ be a listing of all strings $\in\Sigma^*\subseteq L$.
For j=1,2,… run M on $s_1,\ldots,s_j$ for j time-steps.
If M accepts an s, print s. Keep increasing j.

Other Computational Models

We can consider many other ‘reasonable’ models of computation: DNA computing, neural networks, quantum computing…

Experience teaches us that every such model can be simulated by a Turing machine.

Church-Turing Thesis:

The intuitive notion of computing and algorithms is captured by the Turing machine model.

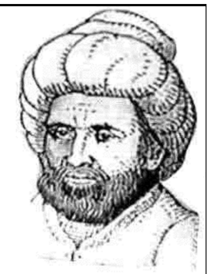
Importance of the Church-Turing Thesis

The Church-Turing thesis marks the end of a long sequence of developments that concern the notions of “way-of-calculating”, “procedure”, “solving”, “algorithm”.

Goes back to Euclid’s GCD algorithm (300 BC).

For a long time, this was an implicit notion that defied proper analysis.

“Algorithm”



After Abū ‘Abd Allāh Muhammed ibn Mūsā al-Khwārizmī (770 – 840)

His “Al-Khwarizmi on the Hindu Art of Reckoning” describes the decimal system (with zero), and gives methods for calculating square roots and other expressions.

“Algebra” is named after an earlier book.

Hilbert’s 10th Problem

In 1900, David Hilbert (1862–1943) proposed his *Mathematical Problems* (23 of them).

The Hilbert’s 10th problem is: **Determination of the solvability of a Diophantine equation.**
Given a Diophantine equation with any number of unknown quantities and with integer coefficients: *To devise a process according to which it can be determined by a finite number of operations whether the equation is solvable in integers.*

Diophantine Equations

Let $P(x_1,\ldots,x_k)$ be a polynomial in k variables with integral coefficients. Does P have an integral root $(x_1,\ldots,x_k) \in Z^k$?

Example: $P(x,y,z) = 6x^3yz + 3xy^2 – x^3 – 10$ has integral root $(x,y,z) = (5,3,0)$.

Other example: $P(x,y) = 21x^2 – 81xy + 1$ does not have an integral root.

(Un)solving Hilbert's 10th

Hilbert's *"…a process according to which it can be determined by a finite number of operations…"* needed to be defined in a proper way.

This was done in 1936 by Church and Turing.

The impossibility of such a process for exponential equations was shown by Davis, Putnam and Robinson.

Matijasevič proved that Hilbert's 10th problem is unsolvable in 1970.

Describing TM Programs

Three Levels of Describing algorithms:
- formal (state diagrams, CFGs, et cetera)
- implementation (pseudo-code)
- high-level (coherent and clear English)

Describing input/output format:
TM's allow only strings $\in\Sigma^*$ as input/output.
If our X and Y are of another form (graph, Turing machine, polynomial), then we use <X,Y> to denote 'some kind of encoding $\in\Sigma^*$'.